Tratando múltiplos casos de teste

Henrique Pinto, Setembro de 2012

Em geral, quando você submete uma solução para um problema, ela é testada em vários casos de teste, e não apenas em um. Em algumas competições, isso é feito executando seu programa uma vez para cada caso de teste, mas essas competições são raras. Em geral, seu programa deve ser capaz de tratar múltiplos casos de teste em apenas uma execução.

Cada problema especifica a forma em que os casos de teste são dados. Há três opções comuns:

- O enunciado especifica o número de casos de teste, ou a entrada começa com esse número;
- O fim da entrada é indicado por um valor especial;
- O fim da entrada é indicado pelo fim do arquivo.

Vamos ver como tratar cada um desses casos.

Problemas onde o número de casos de teste é fornecido

É bem raro que o número de casos de teste seja completamente fixo, e dado no problema. Mas isso acontece. Se o enunciado disser que haverão, digamos, 10 casos de teste, seu programa deve ser construído de forma a tratar todos eles. Por exemplo, usando um `for`:

```
for (int i = 0; i < 10; ++i) {
    entrada = ler_um_caso_de_teste();
    saida = processar_caso_de_teste(entrada);
    std::cout << saida << '\n';
}
```

O caso mais comum para problemas desse tipo, porém, é que o número de casos de teste *não* seja fixo, e sim seja dado como a primeira informação disponível na entrada. Um exemplo é o problema Braceletes Mágicos, onde o número de casos de teste é dado na primeira linha da entrada. Assim como no caso de problemas com número fixo de casos de teste, basta usar um `for` para tratar todos eles. A única diferença é que, nesse caso, o limite de iteração do `for` não é uma constante, mas uma variável. Por exemplo, uma forma de processar a entrada para o Braceletes Mágicos é:

```
int t;
string bracelete, proibida;
```


Notícias

Seletiva Interna para a Maratona de Programação
11 Aug 2013
Inscrições Abertas

Leia mais »

Seletiva Interna para a Maratona Mineira 2013
21 Apr 2013
11 times da UFMG e 21 times externos participaram; confira os resultados.

Leia mais »

Primeira Maratona Mineira de Programação
01 Jun 2012
Times da UFMG conquistam primeiro, segundo e nono lugares na competição

Leia mais »

Próximas Competições

Maratona Mineira de Programação
25 de Maio
Itajubá, MG

```
    std::cin >> proibida >> bracelete;
    bool existe = resolve(proibida, bracelete);
    std::cout << (existe? 'S' : 'N') << '\n';
}
```

Problemas onde a entrada termina com um valor especial (sentinela)

Problemas desse tipo são bastante comuns em competições. Em vez de o número de casos de teste ser dado de forma explícita, você deve ler casos da entrada até encontrar um valor especial, descrito no enunciado do problema, que sinaliza o fim da entrada. Um exemplo de problema desse tipo é o f91. Nesse problema, como descrito no enunciado, cada caso de teste é dado por uma linha contendo um único inteiro, e a entrada termina quando o inteiro lido for igual a zero. Note que o valor que indica o fim da entrada (zero, nesse exemplo) *não* é um caso de teste, e não deve ser tratado como um.

Como o número de casos de teste não é conhecido a priori, não é possível determinar um limite para o número de iterações. Uma forma comum de tratar esse tipo de problema é ter um loop infinito, e um teste dentro do loop que verifica a condição de parada. Por exemplo, no caso do `f91`, poderíamos ler a entrada da seguinte forma:

```
int n;
while (true) {
    cin >> n;
    if (n == 0)
        break; // Fim da entrada encontrado, sai do loop
    printf("f91(%d) = %d\n", n, f91(n));
}
```

Problemas onde a entrada termina com o fim do arquivo

Há também problemas onde o número de casos não é dado, e nem há um valor sentinela que indique o fim da entrada. Nesses casos, você deve ler todos os casos de teste até o fim do arquivo. Um exemplo é o problema Quem vai ser reprovado, cujo enunciado especifica claramente que “A entrada termina com final de arquivo.”

Assim como os problemas em que o fim da entrada é indicado por um valor sentinela, nesses problemas é impossível definir um número de iterações a priori. Logo, usar um loop infinito e uma condição de parada também é uma idéia interessante. Basta alterar a condição de parada para algo que teste se o fim do arquivo foi alcançado.

Se você estiver usando `std::cin` para ler a entrada, avaliar `cin` em contexto booleano retorna `false` se o fim do arquivo

```
while (true) {
    std::cin >> entrada;
    if (!std::cin)
        break; // fim de arquivo atingido
    std::cout << processa_caso_de_teste(entrada) << '\n';
}
```

Note que você tem que testar se chegou ao fim do arquivo apenas **depois** de tentar ler um caso de teste extra. Uma fonte de erros muito, muito, muito comum em competições é testar essa condição antes de tentar ler um caso de teste:

```
while (true) {
    if (!std::cin)
        break; // Não funciona como você esperaria que funcionasse!
    std::cin >> entrada;
    std::cout << processa_caso_de_teste(entrada) << '\n';
}
```

Isso acontece porque, mesmo que você já tenha lido todos os casos de teste do arquivo de entrada, o `cin` (ou o `stdin`, se estiver usando as funções de leitura de C) só vai saber que o fim do arquivo foi alcançado quando você tentar ler alguma coisa e não conseguir. **Lembre-se disso, é importante.**

Se você estiver usando as funções de leitura de entrada de C, a verificação é semelhante. Você pode usar a função `feof` para testar o fim do arquivo:

```
while (true) {
    scanf("%d", &entrada);
    if (feof(stdin))
        break;

    printf("%d\n", processa_caso_de_teste(entrada));
}
```

Alternativamente, você pode usar o valor de retorno de `scanf` diretamente: a função retorna `EOF` se o fim do arquivo foi atingido:

```
while (scanf("%d", &entrada) != EOF) {
```